PadhAI: Recurrent Neural Networks

One Fourth Labs

Setting the context

We are going to learn about Recurrent Neural Networks through 6 jars of machine learning, namely **Data, Tasks, Model, Loss Function, Learning Algorithm** and **Evaluation.**

Data and Tasks

This jar undertakes
1. How we are going to represent character and word data as numbers.
2. RNNs are used for 3 types of tasks:

A. *Sequence Classification (Sentiment Classification, Video Classification)*

Only one output is produced in the end, the output depends on the whole input sequence.

Therefore, for **n number** of words in the sequence model generates **only one** output**.**

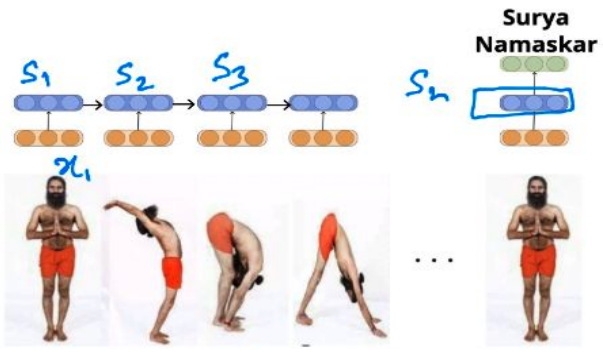


Fig.: Video Classification

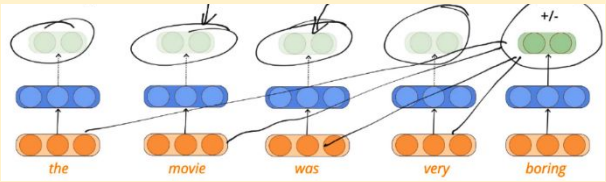


Fig.: Sentiment Classification

How to Pre-process input data?

Input data

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
The	first	half	was	very	boring	.	
Great	performance	by	all	the	lead	actors	.
The	visual	effects	were	stunning	.		
The	movie	was	a	waste	of	time	.

label

y
0
1
1
0

As we can see that the length of every input is not the same, we need to pre-process our data before we train our model.

Steps to Pre-process data.

Step 1: Making the sequences of the same length.

- Define special symbols: <sos>, <eos>, <pad>
- Prepend/Append <sos>, <eos> to each sequence
- Find maximum input length across all sequences (say, 10)
- Add special word <pad> to all shorter sequences so that they become of the same length (10, in this case)

$< sos > \ - \ Start\ of\ sequence$

$< eos > \ - \ End\ of\ sequence$

$< pad > \ - \ Artificial\ word$

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
<sos>	The	first	half	was	very	boring	.	<eos>	<pad>
<sos>	Great	performance	by	all	the	lead	actors	.	<eos>
<sos>	The	visual	effects	were	stunning	.	<eos>	<pad>	<pad>
<sos>	The	movie	was	a	waste	of	time	.	<eos>

y
0
1
1
0

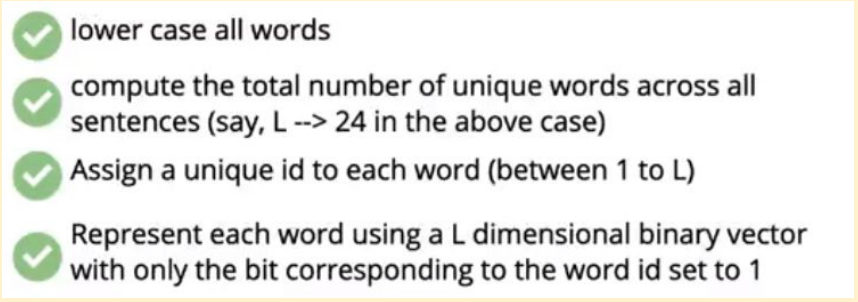
< sos > marks the starting point of the sentence/sequence.

<eos> marks the ending point of the sentence/sequence.

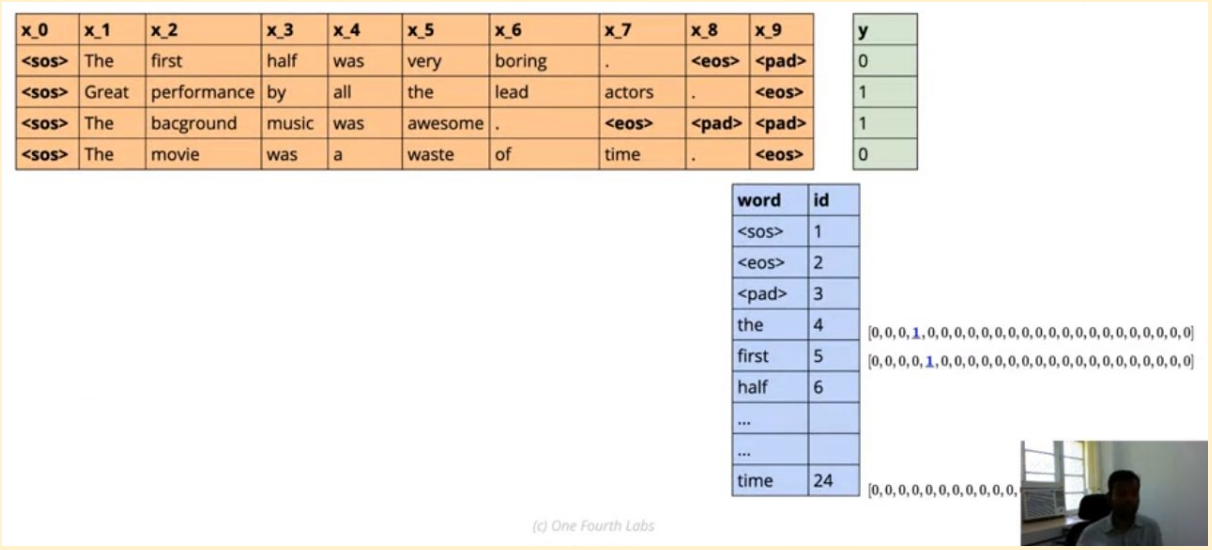
<pad> is an artificial word, used to fill all the left empty spaces (after <eos>) to make the input sentence of the same length as that of the longest sentence/sequence in the dataset.

After appending <sos>, <eos> and <pad> to the sentences, all the sentences become of the same length (10). Now the dataset is a matrix of dimensions 4 × 10.

Step 2: Converting words into numbers.



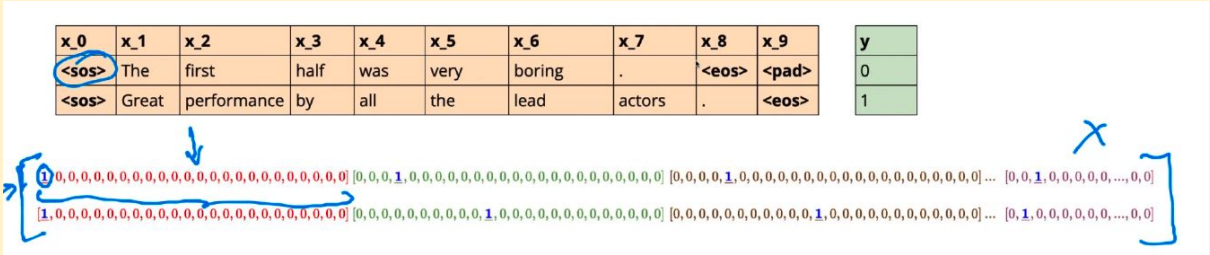
Create one hot vectors for every unique word in the dataset.



The blue table is a collection of all the unique words that are there in the dataset and for every word there's a unique index associated to it.

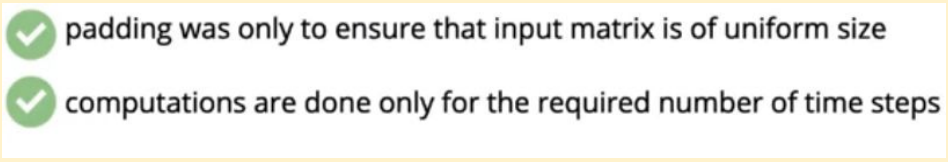
- **First three indices** are always assigned to **<sos>, <eos> and <pad>** respectively.
- In our example there are 24 unique words including <sos>, <eos> and <pad>.
- For **every unique word** a **unique 24-dimensional one hot vector** is created, according to the **index assigned to each word**. In each 24-dimensional one hot vector **only the bit corresponding to the word will be ON** and **rest of the 23 bits will be OFF.**

After assigning vectors, below is how our dataset looks like

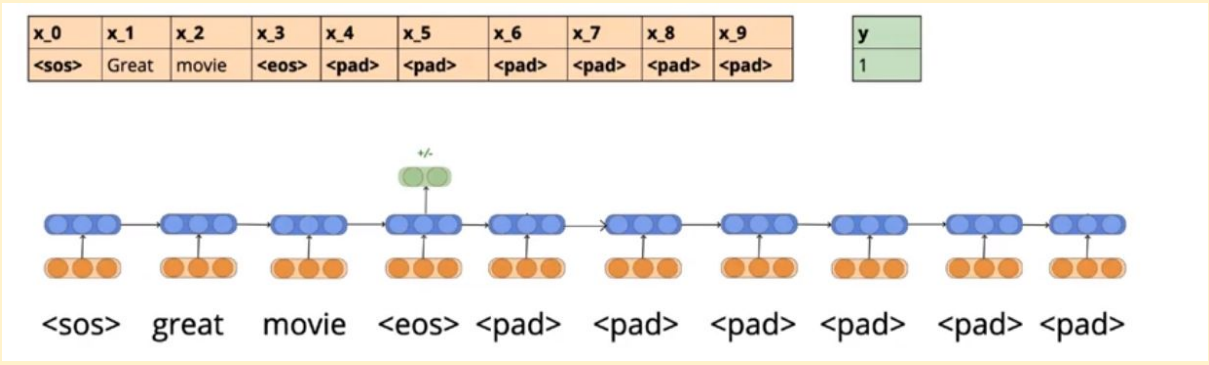


The dataset is now a matrix in which each word and special symbol is represented using their corresponding one hot vectors. Model gets trained using these one hot vectors.

A clarification about padding.



Consider the sentence **“Great movie”** model needs to output the result where it encounters **<eos>** special symbol.



In addition to the dataset we also feed in a **vector containing the true lengths of the sentences (including <sos> and <eos>).** In Accordance to the **length vector Pytorch will do computations only till** s_l **, where *l*** is the length of the sentence**.**

a. *Sequence Labelling (Part of Speech tagging, Named Entity Recognition)*

For every word in the sequence model needs to attach a label to each word.
Therefore, for **n number** of words in the sequence model generates **n number** of outputs**.**

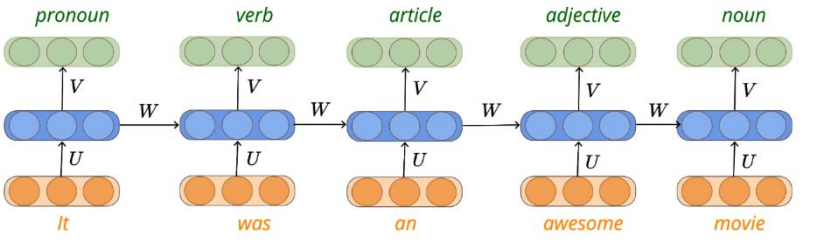


Fig.: Part of speech tagging.

For **Named Entity recognition** consider this example

"Ram went to Delhi via Chandigarh yesterday."

In the above sentence model has to label the words as **Named Entity** or **Not Named Entity.**

- **Ram**, **Delhi** and **Chandigarh** are **named entities**.
- **went, to, via** and **yesterday** are **not named entities.**

How to pre-process the data?

Input data

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
The	first	half	was	very	boring	.	
Great	performance	by	all	the	lead	actors	.
The	bacground	music	was	awesome	.		
The	movie	was	a	waste	of	time	.

Sentences are not of the same lengths.

labels corresponding to each input

y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
DT	AJ	NN	VB	JJ	JJ	PC	
AJ	NN	PP	PN	DT	JJ	NN	PC
DT	JJ	BB	VB	JJ	PC		
DT	NN	VB	DT	NN	PP	NN	PC

labels are not of the same lengths.

✅ Words need to be converted to numbers

✅ Labels/outputs need to be converted to numbers

Steps to pre-process data.

Step 1: Making the dataset of equal length.

- Define special symbols/**labels**: <sos>, <eos>, <pad>
- Prepend/Append <sos>, <eos> to each input/**output** sequence
- Find maximum input length across all sequences (say, 10)
- Add special word/**label** <pad> to all shorter sequences so that they become of the same length (10, in this case)

This time append **<sos>, <eos>** and **<pad>** to not only **sentences** but to **labels** as well.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
<sos>	The	first	half	was	very	boring	.	**<eos>**	**<pad>**
<sos>	Great	performance	by	all	the	lead	actors	.	**<eos>**
<sos>	The	bacground	music	was	awesome	.	**<eos>**	**<pad>**	**<pad>**
<sos>	The	movie	was	a	waste	of	time	.	**<eos>**

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
<sos>	DT	AJ	NN	VB	JJ	JJ	PC	**<eos>**	**<pad>**
<so>	AJ	NN	PP	PN	DT	JJ	NN	PC	**<eos>**
<sos>	DT	JJ	BB	VB	JJ	PC	**<eos>**	**<pad>**	**<pad>**
<sos>	DT	NN	VB	DT	NN	PP	NN	PC	<eos>

Step 2: Converting words and labels into numbers.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
<sos>	The	first	half	was	very	boring	.	**<eos>**	**<pad>**
<sos>	Great	performance	by	all	the	lead	actors	.	**<eos>**
<sos>	The	bacground	music	was	awesome	.	**<eos>**	**<pad>**	**<pad>**
<sos>	The	movie	was	a	waste	of	time	.	**<eos>**

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
<sos>	DT	JJ	NN	VB	JJ	JJ	PC	**<eos>**	**<pad>**
<so>	JJ	NN	PP	PN	DT	JJ	NN	PC	**<eos>**
<sos>	DT	JJ	NN	VB	JJ	PC	**<eos>**	**<pad>**	**<pad>**
<sos>	DT	NN	VB	DT	NN	PP	NN	PC	<eos>

word	id	
<sos>	1	
<eos>	2	
<pad>	3	
the	4	$[0,0,0,\underline{1},0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$
first	5	$[0,0,0,0,\underline{1},0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$
half	6	
...		
...		
time	24	$[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\underline{1}]$

label	id	
<sos>	1	
<eos>	2	
<pad>	3	
DT	4	$[0,0,0,\underline{1},0,0,0,0,0,0]$
JJ	5	$[0,0,0,0,\underline{1},0,0,0,0,0]$
NN	6	
...		
...		
PN	10	$[0,0,0,0,0,0,0,0,0,\underline{1}]$

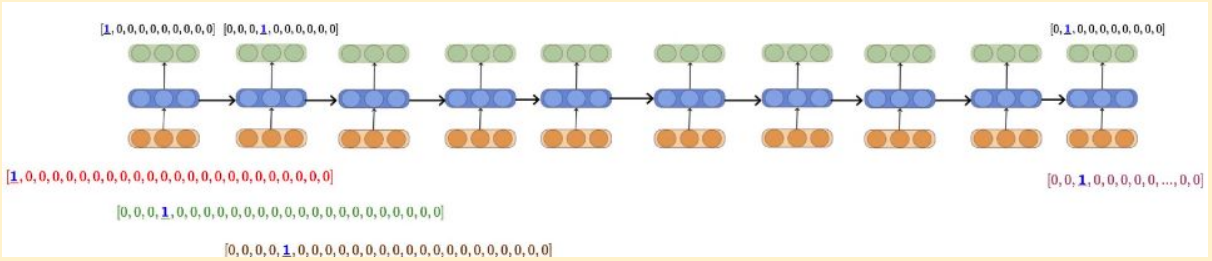
The blue table is a collection of all the unique words that are there in the dataset and for every word there's a unique index associated to it.

First three indices are always assigned to **<sos>, <eos> and <pad>** respectively.
In our example there are 24 unique words including <sos>, <eos> and <pad>.
For **every unique word** a **unique 24-dimensional one hot vector** is created, according to the **index assigned to each word**. In each 24-dimensional one hot vector **only the bit corresponding to the word will be ON** and **rest of the 23 bits will be OFF.**

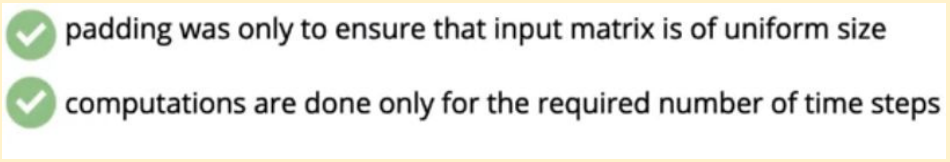
Similarly the purple table is a collection of all the unique labels that are there in the dataset and for every label there's a unique index associated to it.

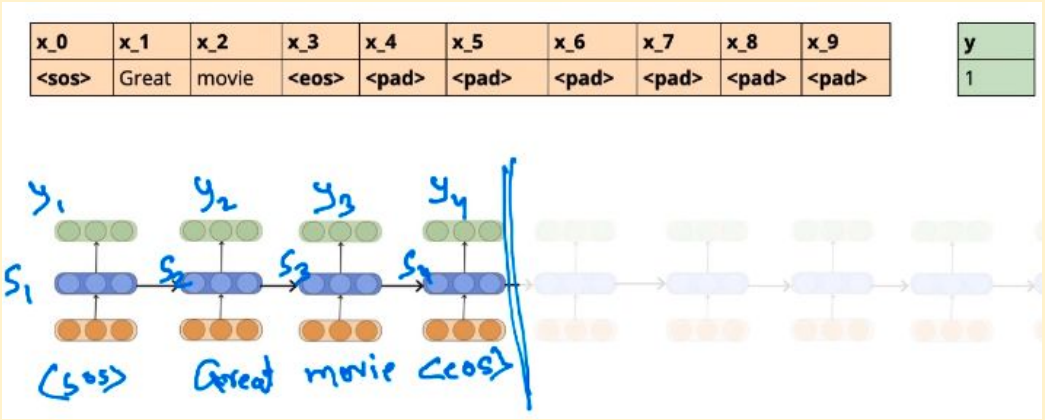
First three indices are always assigned to **<sos>, <eos> and <pad>** respectively.
In our example there are **10 unique labels** including **<sos>, <eos> and <pad>.**
For **every unique label** a **unique 10-dimensional one hot vector** is created, according to the **index assigned to each label**. In each **10-dimensional** one hot vector **only the bit corresponding to the label will be ON** and **rest of the 9 bits will be OFF.**

After assigning the vectors to words and labels two matrices are created containing the vectors corresponding to the words and labels respectively. Now these matrices will used as our dataset to train our model.



A clarification about padding.





x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
<sos>	Great	movie	<eos>	<pad>	<pad>	<pad>	<pad>	<pad>	<pad>

y
1

In addition to the dataset we also feed in a **vector containing the true lengths of the sentences (including <sos> and <eos>).** In Accordance to the **length vector Pytorch will do computations only till** s_l **, where *l*** is the length of the sentence**.**

b. **Sequence Generation** *(Machine Translation, Transliteration)*

In sequence generation, for our **input (a sequence)** an **output (a sequence)** is generated and both the sequences may be of **different lengths**.

For example: **Translation of an English sentence to a Hindi sentence.**

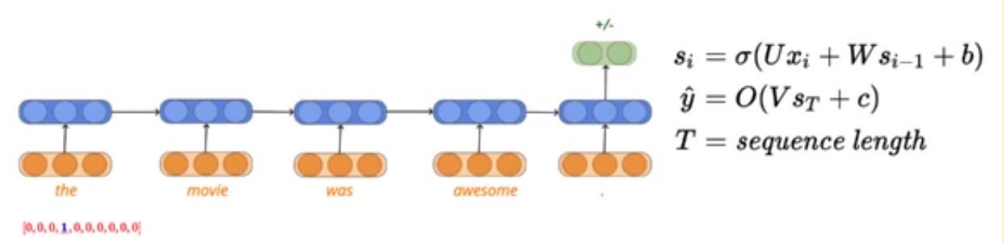
I am going home

मैं घर जा रहा हूं

Observe that the length of generated output sentence (5 words) is different from the length of input sentence (4 words).

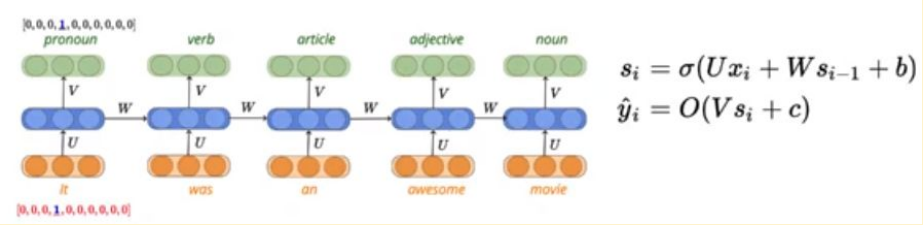
Model:

Sequence Classification



- Output $\hat{y}$ is calculated at last time step.
- s_T is the value of state at last time step.

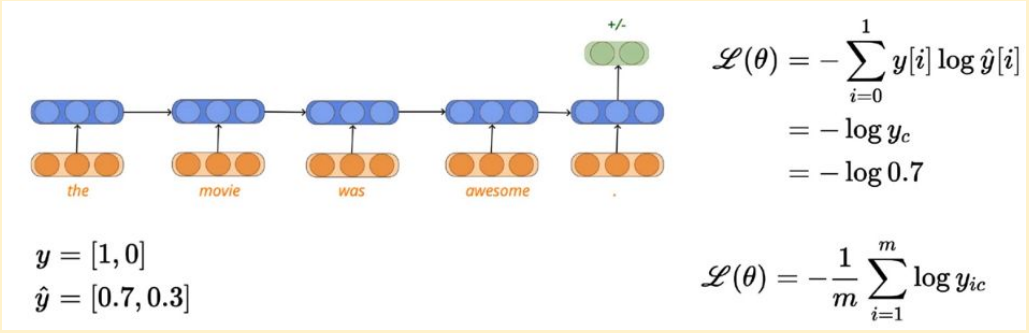
Sequence Labelling



- Output $\hat{y}_i$ is calculated at each time step.
- s_i is the value of state at i^{th} time step.

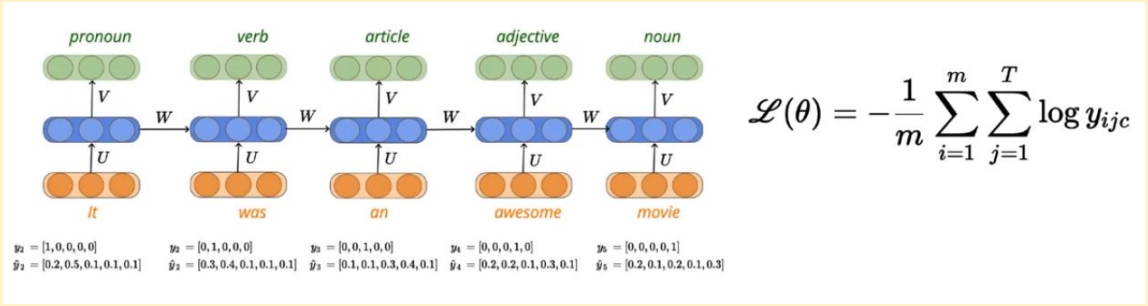
Loss Function:

Loss Functions in Sequence Classification tasks



- y is true output.
- $\hat{y}$ is the predicted output.
- m is the total number of training examples.

Loss Functions for Sequence Labelling tasks.



Here, ***i*** is used to represent **training example** and ***j*** is used to represent the **time step**, **m** represents total number of **training examples** & **T** represents total number of **time steps**.

- y is the true output.
- $\hat{y}$ is the predicted output.

Learning Algorithm:

Initialise w, b

Iterate over data:

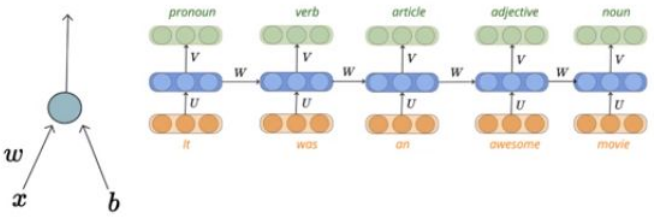
$compute\ \hat{y}$

$compute\ \mathscr{L}(w, b)$

$w = w - \eta\Delta w$

$b = b - \eta\Delta b$

till satisfied



$Earlier : w, b$

$Now : w_{11}, w_{12}, ..., u_{11}, u_{12}, ..., v_{11}, v_{12}$

$Earlier : L(w, b)$

$Now : L(W, U, V)$

Initialise w, b

Iterate over data:

$compute\ \hat{y}$

$compute\ \mathscr{L}(w, b)$

$w_{11} = w_{11} - \eta\Delta w_{11}$

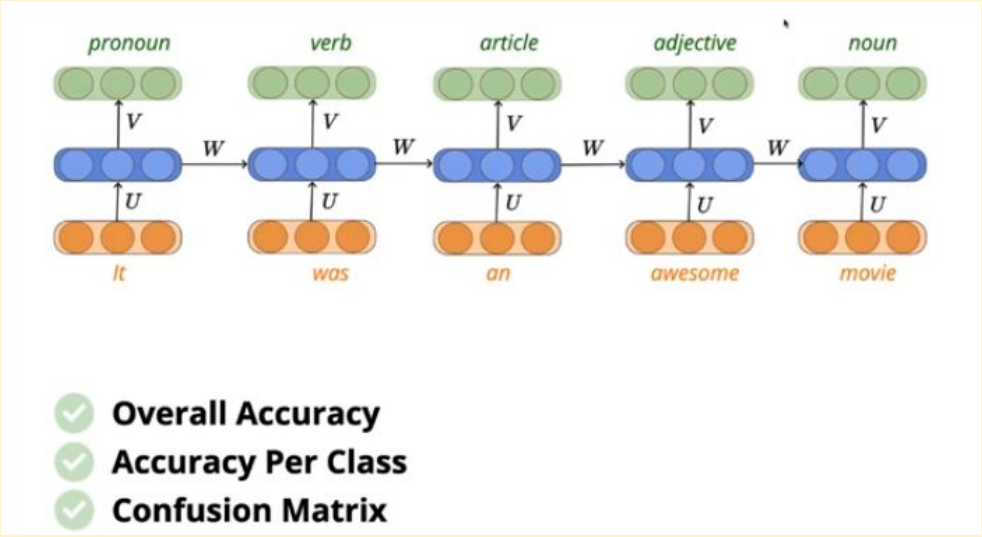
$u_{12} = u_{12} - \eta\Delta u_{12}$

....

$v_{13} = v_{13} - \eta\Delta v_{13}$

till satisfied

Evaluation:



Overall accuracy:

$$\frac{Total\ number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Accuracy Per Class: We calculate accuracy for each class separately. This tells us how many times we have hit the correct prediction for each class. For class with least accuracy, we need to train our model with a greater number of training examples for that class. This helps us to precisely know where our model is generating high loss.

Confusion matrix:

You not only get to know **per class accuracy** of the model but also where the model is getting confused between two or more classes.